TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SIAMAK RAHIMI MOTEM

BOOKMARKING AND SEEKING TOOL FOR ONLINE VIDEOS

Master of Science thesis

Examiner: prof. Kari Systä
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 4th May 2016

ABSTRACT

SIAMAK RAHIMI MOTEM: BOOKMARKING AND SEEKING TOOL FOR ONLINE VIDEOS
Tampere University of technology
Master of Science Thesis, 45 pages
May 2016
Master's Degree Program in Information Technology
Major: Pervasive Computing
Examiner: Professor Kari Systä

Keywords: Bookmarking Video, Video Summarization, HLS, Efficient Seeking, Adaptive Streaming

At 2014, 66% of internet traffic was related to video content [1]. This number and everyday experience shows that by improving handheld device capabilities, social networks and internet speed, the video content which has been seen and posted is taking up most internet traffic. As a result, this thesis focuses on improving the user experience with videos in two supplementary features: Bookmarking videos and enhanced seeking.

There have been many cases, such as CCTV and medical cases, where making a video summary and video synopsis does not serve the purpose and the whole video must be available. However, the user is only interested in certain moments in the video. Usually in these cases either a video summary is generated along the main video, interesting moments in the video is kept as a note, or the user finds it manually by making seeking forward and backward.

Video bookmarking, which means keeping the original video and makes a list of interesting moments in the video, so that the user can seek toward them by selecting them solves this issue. The bookmarks are standardized JSON objects in a JSON array that can be added, deleted or modified. In their simplest form, they have a relative start time, duration and a caption.

Having bookmarks available in the cases mentioned above, user behavior can be predicted. The user is highly likely to request a seek for a bookmarked moment. By caching the video content, which has the bookmarked content, the user does not need to wait for buffering to see the video playing from the seek target. Currently, the user must wait for buffering. It has a major impact in cases such as CCTV and medical cases, where different cameras have recorded a scene from different angles and a seek action must seek all the video content, at the same time.

In this thesis, an application has been developed as proof of concept which has met both requirements. It has been developed over an existing application, which is used for treatment of epilepsy by using automated seizure detection.

PREFACE

The current thesis has been done as a project at Intopalo for Neuro Event Labs company known as NEL under a joint supervision of NEL and TUT. Application development wise, it has started at January 2015 and finished at June 2016.

First and foremost, I would like to thank my supervisor, Kari Systä, who supported me during my career and while writing the thesis. His flexibility, understanding, trust and important feedbacks have helped me a lot to make this happen.

I would also like to thank my Intopalo NEL support for doing my thesis. Kimmo Eklund supervision. Juha Latvala a perfect CEO of Intopalo and Jussi Piispanen a co-founder of NEL supports and the great support I got from my team members and cell leads Samuel Nevala and Andrew Knight.

I would like to thank my friends, Mehdi Peyvandi, Saeed Mehrang, Mohsen Razzaghi and Mohammad Shariat who supported me with my thesis.

I would like to specially thank my family. Aziz, Nina and Sina. My parents have always had a non-stop support and love. They have taught me what does love and support really mean. My brother who is my best friend and supporter since childhood, has helped me a lot to keep working on the thesis. I have enjoyed every moment of my life being with them.

Finally, I would like to thank Mana Deljoo who has made my life perfect and complete with her love. Mana's love and support has kept me motivated and helped me to finalize the thesis.

Tampere, 21.5.2017

Siamak Rahimi Motem

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AES	Advanced Encryption Standard
AVC	Advanced Video Coding
CAGR	compound annual growth rate
CCTV	Closed-Circuit Television
CDN	Content Delivery Network
CPU	Central Processing Unit
DASH	Dynamic Adaptive Streaming over HTTP
DRM	Digital Rights Management
HD	High Definition
HDS	HTTP Dynamic Streaming
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
HLS	HTTP Live Streaming
JSON	JavaScript Object Notation
LSS	Live Smooth Streaming
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
R&D	Research and Development
RTSP	Real Time Streaming Protocol
SD	Standard Definition
SVC	Scalable Video Coding
TCP	Transmission Control Protocol
TUT	Tampere University of Technology
UDP	User Datagram Protocol
VNI	Visual Network Index
VOD	Video on Demand
URL	Uniform Resource Locator
UTC	Universal Time Clock

1. INTRODUCTION

1.1 Introduction

Looking at the first web pages and initial years of computer usage and comparing it with nowadays, as time has passed, images and afterwards videos started to be used more. Improved CPU capabilities, the rapid spread of smartphones, internet speed increase in both mobile and desktop platforms are some reasons for this phenomenon. Cisco VNI has stated that internet video streaming and downloads are beginning to take a larger share of bandwidth and they have forecasted that by 2020, more than 80% of internet traffic will be internet video. Currently, it is 71% and its CAGR is forecasted 31% in 2015-2020[2].

Figure 1 shows how video share in mobile data traffic is increasing.

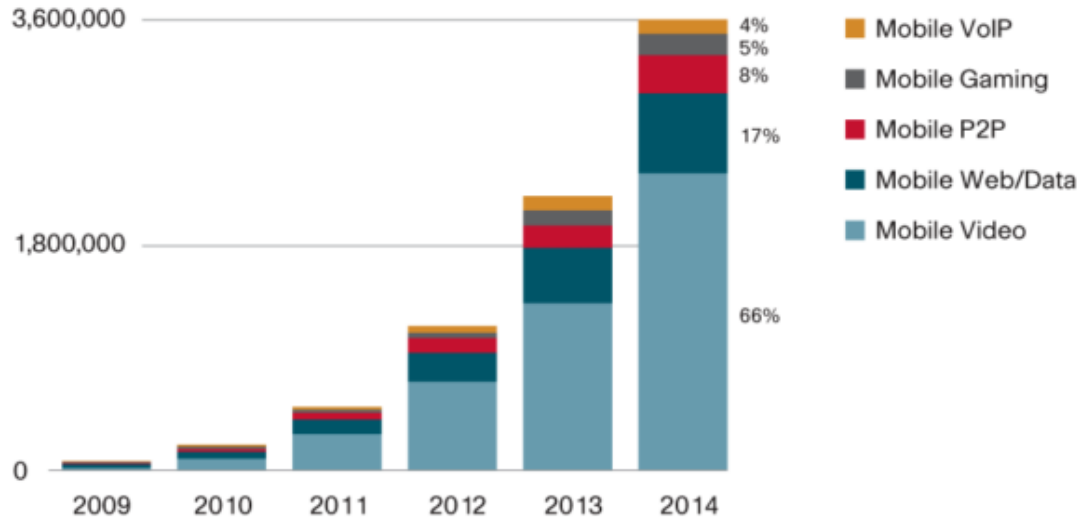

Figure 1. Video will account for 66 percent of global mobile data traffic by 2014[1]

It is apparent that video as a media tool is more descriptive, useful and interesting in many cases in comparison with text, image or audio media files, but the main hindrance to use it more in comparison with other media files is the fact that video files are bigger in size. Delivering video content to the end user requires more internet bandwidth and this is the reason why there are many ways to compress video files and different methods to deliver them.

There are two main methods for accessing online videos: downloading them and streaming them online. The client downloads the video file which exists on a remote

server and saves it on a computer or network media player's hard drive. Thereafter, the user can access the video file anytime even if there is no internet connection. On the other hand, "Streaming" describes the act of playing media on one device when the media is saved on another. The file does not need to be moved or copied to the device that is playing it.

Although downloading a video file enables the user to access and play the file several times, video files are big in size and it requires more hard disk space to save the file rather than watch it online. In streaming, accessing to video files requires less local space, but it requires access to the network to buffer the needed amount of data to start playing, which depends on the implementation method used, and this translates to some amount of waiting time depending on the internet speed and video quality. In most cases replaying it, requires getting data from the network again.

Other than different compression methods to make the video files smaller in size, there have been cases where a user has recorded a long video clip and he or she is interested in only some parts of it. Video summarization means making a subset of a main video in which only the interesting parts of the longer video clip exists.

There are two ways to summarize a video. 1. Manually, by using a software which allows a user to cut the video and concatenate different video chunks together, such as Adobe premiere 2. Automatically, by using computer vision and automatically detecting the interesting part. Video synopsis is an example of such techniques[3].

However, in some cases cutting the video comes with the cost of losing data and it is not favorable in medical cases. For a neurologist, what happens before the seizure is valuable information for some types of seizures and can affect the diagnosis. In a similar case storing the whole video is needed although we are only interested in some parts of it.

1.2 Purpose of the Thesis

The problem to be addressed in this thesis is the case in which there is a video file that must be streamed, and it cannot be summarized because loss of content cannot be tolerated by the user, and the end-user is more interested in some parts of the video than others.

This can happen, for example in medical cases where the video cannot be downloaded due to the patient's privacy and the high video quality is used to get more data to provide a better diagnosis.

1.3 Thesis Structure

The thesis contains 5 chapters. First chapter is the introduction. It defines the problem which this thesis solves. Chapter 2 explains the current methods of video summarizing and streaming. Chapter 3 provides the design and implementation details of the solution. Different use cases which can be achieved by the generic solution are also stated in this chapter. In Chapter 4 results of the proof-of-concept application are reported and evaluated. Chapter 5 concludes the thesis.

2. BACKGROUND

2.1 Video Summarizing

When servers started to get filled with lengthy videos such as video clips recorded from surveillance cameras, the need to store only the important parts of the video became apparent. By definition, video summarization methods attempt to abstract the main occurrences, scenes, or objects in a clip in order to provide an easily interpreted synopsis [4].

2.1.1 Use Cases

The generic use case for making summary of a video is when we have a video which we are not interested in having to go through all of it. In other words, we are only interested in some parts of the video. Some examples of these cases are sport videos, music videos, traffic videos, surveillance videos [5] etc. It can help us produce movie highlights or documentaries. Finding the movement of an insect in a long video is a concrete example of how it can help us produce documentaries faster. In addition, in medical cases, there has been cases such as epilepsy treatment in which the neurologist needs to see patient seizures to diagnose, and prescribe treatment based on that.

2.1.2 Techniques

Other than making a video summary manually which is perfect but time-consuming, there has been variety of ways to automate this process. Ajmal et al. have explained different techniques in details, and have compared them against each other[5]. The hierarchy of different video summarization techniques is depicted in figure 2. Based on the context in which a user needs to perform video summarization, the best technique may vary. Some of these techniques use machine learning and feedback loops in both supervised and unsupervised learning methods to get acceptable results. In supervised learning, there should be adequate data covering most of the interesting scenes, to train the machine.

In some cases, a standalone technique is not enough and a combination of techniques should be used. For example, in a medical case both gesture and audio video based techniques should be combined, because neither of them can be used standalone without loss of content.

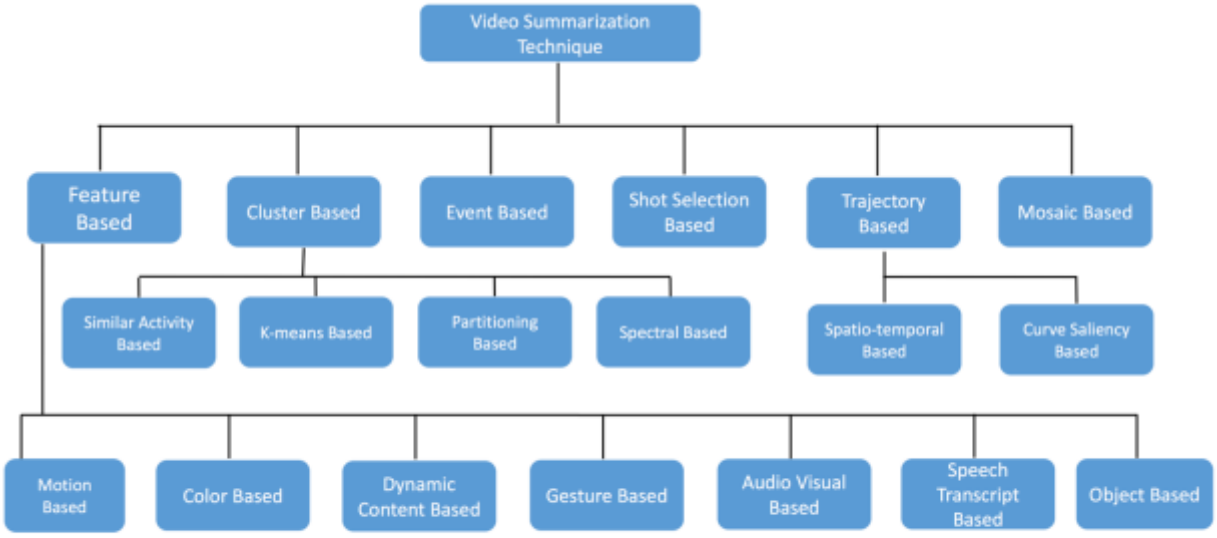

Figure2. Hierarchical structure of video summarization techniques[5]

2.1.3 Problems

The output of a video summarization is a shorter video. Ideally it should cover all the interesting subsets of the video. However, depending on the case, we can obtain a desired level of accuracy. Detecting a movement can be done with a higher accuracy rate than detecting the seizure of an epileptic patient. Monireh et al. have compared the accuracy rate of different techniques. The fact that there are several ways of video summarization denotes that this is an evolving field, and that there is still no final solution for it. However, it has gone beyond R&D and although it is not perfect in some areas now, it is being used in industry. Companies such as BriefCam, IntelliVision, etc. are offering services in this field.

The main problem of making a summary using a technique which is not fully accurate is the **potential loss of content**. In some cases, such as CCTV the full data is stored for a certain amount of time and if nothing is reported it gets discarded. However, in medical cases such as epilepsy treatment the reason to perform summarization is to save doctor's time and loss of content means less efficient diagnoses. It is also important to know what happened before the seizure or after it. In development of a video summarization technique, there is also a need for the algorithm developer to go through the video and see which time spans has been detected by the algorithm to fine tune sensitivity.

2.2 Streaming Video

Streaming video means sending video content from the internet and playing it in a client media player without needing to download the whole file. Streaming can be done live or on demand also known as VOD.

In video streaming, usually encoded video is served by the server. Encoding helps to reduce the raw video file in either a lossy or lossless compression. Users can observe video clips through the network. Clients require having a corresponding decoder to decode the video and let the end user watch it.

A typical video streaming system is depicted in the figure below.

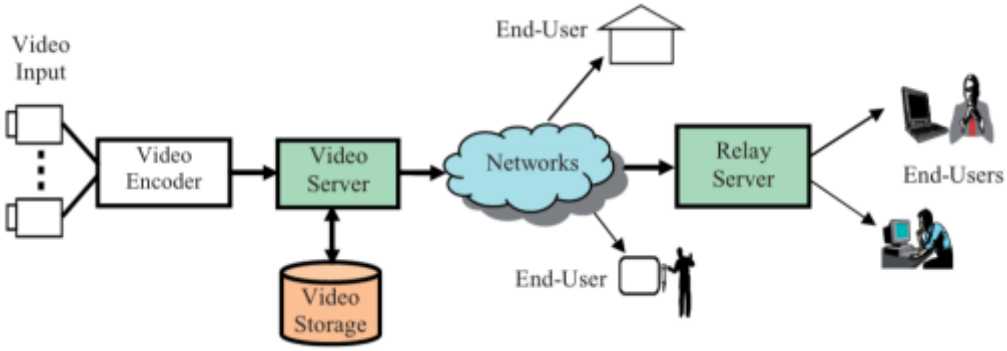

Figure 3. Typical Video Streaming System[6]

Cahralampos et al[7] explains four possible ways to accomplish access media content over the web. They include: download and play, traditional streaming, progressive download and HTTP adaptive streaming.

2.2.1 Download and Play

It is the simplest and oldest way to watch videos. In this method, the video file is downloaded in its entirety. Then it can be played from the local environment by a player. The player needs to have the decoder to decode the file. This method still has its own specific use cases. For short videos sent to messenger apps such as Telegram, this technique is still being used. However, in most cases which the video file is large, it is not answering market needs. In addition, handling copyright and DRM is also harder in this method.

2.2.2 Traditional Streaming (RTSP)

Real Time Steaming Protocol (RTSP) is a network control protocol. It is designed to establish and control either a single, or several, time-synchronized streams of continuous media such as audio and video [8] Unlike HTTP, RTSP is stateful. It can be

implemented over TCP or UDP. UDP is used when less end-to-end delay is needed and loss of content during transmission is tolerable by the user or the upper level protocols. Client can use basic methods such as Play, Pause, Record, etc. to control the behavior. RTSP can serve both live content and on-demand content. RTSP is designed to get through firewalls but not all the firewalls handle or allow RTSP based connection. In an RTSP connection both client and server can send issue requests [8]. Based on the nature of the content, available acceptable requests may vary. For example, live content does not support seek request.

2.2.3 Progressive Downloading

Progressive download uses HTTP or HTTPS protocol to transfer media content from a server to a client. The key difference between download and play and progressive download is the fact that in Download and Play the user needs to download the whole file and then playback can only start after that. However, in the latter approach, the playback can be started if the minimum needed data is received. The required amount is dictated by the encoder settings and client player settings. Early playback can be done if the container stores the meta-data in the front of the file. This way, the client can acquire all the needed information to perform early playback. Figure 4 shows how early playback can be achieved in progressive download. If the network speed is low and during playback the player does not have the needed buffered data, the player pauses and restarts playback when it has the needed amount. When a user seeks a position, the client starts to download from that position and on having buffered enough, playback starts from there.

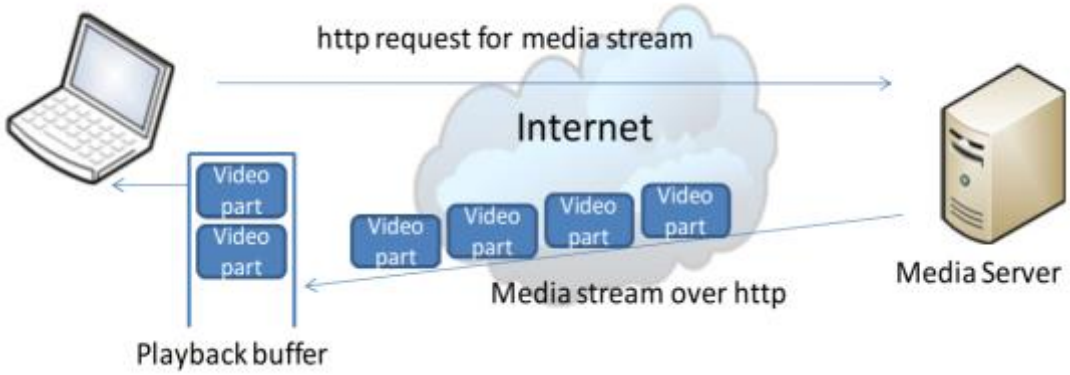

Figure 4. Progressive Download [7]

The problem with this method happens in a scenario when a user opens a page and starts playback of a long video and pauses after a minute but the page remains open. In this scenario, the player client will download all the video content which may never be seen. It means the issuer must pay for transferred data which is not used. YouTube, the most well-known online video center was using progressive download since 2012.

YouTube then stopped using progressive download in favor of HTTP Adaptive streaming.

Progressive Downloading with HTTP does not need special setup of the firewall, because it is already known and accepted protocol unlike RTSP. Although the user experience is similar with streaming, progressive downloading is not exactly streaming because it stores the video on physical drive of the end user. Hence, it is also referred as pseudo-streaming.

2.2.4 Adaptive Bitrate Streaming

Adaptive streaming is a streaming type in which the streaming content changes based on the network condition and CPU capacity. In other words, it adapts the content based on the connection quality and CPU capacity. The purpose is to deliver the best possible user experience. For example, watching a non-interrupting SD video is usually preferred over an interrupting full HD under a poor network condition. However, it is not dictated which bit-rate should be used. As shown in figure 5, players such as YouTube use Auto for adaptive streaming and lets the user manually choose the possible bit rates.

Figure 5. Players let users manually select or let it adapt by choosing auto

For implementing adaptive bitrate streaming there are three ways [9]:

1. Transcoding on request
2. Scalable encoding
3. Stream Switching

In the first approach, the raw content is available on the server side. On receiving incoming request, the controller encodes the media content accordingly. After transcoding it will deliver. Its block diagram is shown in figure 6.

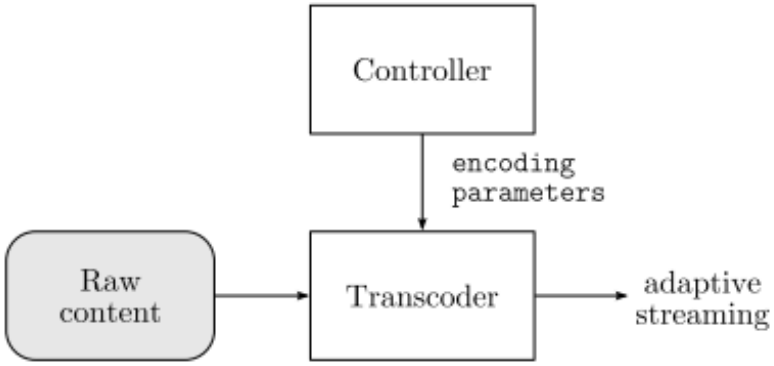

Figure 6. Transcoding approach for adaptive streaming [10]

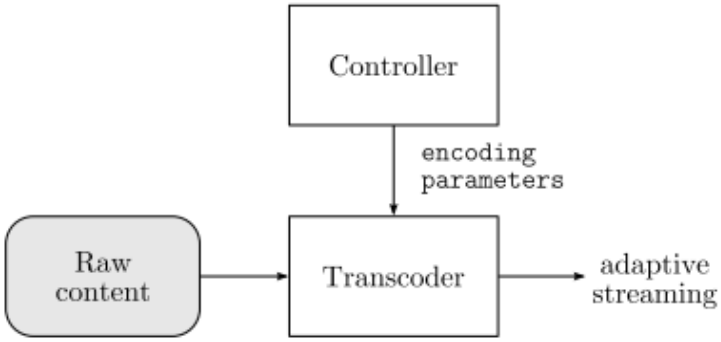

Figure 7. Scalable encoding approach for adaptive streaming

Advantage of this technique is less memory consumption because it stores only one instance of the media content which is the raw content.

The problem of the *transcoding on reques*t is that CPU load is high as encoding is needed for every request even if it has been encoded. Re-encoding is needed because encoded medias are not stored and have only been transmitted.

In *scalable encoding*, the media content is encoded with a scalable codec such as H.264/MPEG-4 AVC prior any requests. H.264/MPEG-4 AVC is an advanced video codec which can perform scalable video coding (SVC). In this technique, the encoding of a high-quality video bit-stream contains one or more subset bit streams. These subsets are derived by dropping packets from the larger video to achieve less required bandwidth. The key point in scalable encoding is the fact that it does not need re-encoding to achieve different bit rates.

The advantage of *scalable encoding* is it only needs to store one file in the server. In comparison with the first approach, in scalable encoding the server needs to encode it only once and re-use it for all incoming requests [11]. It means that the process load is decreased using this technique.

The problem of *scalable encoding* is deployment of it into CDN is complicated because specialized severs are required to implement logic [10]. It is also dependent on the codecs which support SVC. The block diagram of this approach is shown in figure 7.

In *stream switching* raw video is encoded in R versions of different bit-rates and stored in server. These are called video levels. On an incoming request, the controller unit will decide based on a dynamic algorithm which video level to send. Like other techniques if the network bandwidth of the user changes the algorithm seamlessly switches to other video levels. Figure 8 shows a schematic view of stream switching.

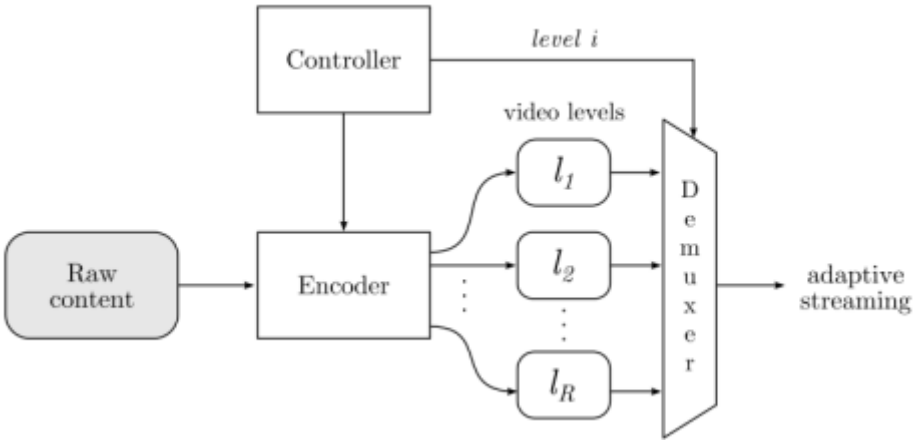

Figure 8. Stream Switching approach for adaptive streaming[9]

Requiring the least processing among different techniques is the advantage of *stream switching*. It also allows the user to freely choose the appropriate codec and does not dictate the codec, unlike *scalable encoding*.

High memory consumption in this technique reduces processing time. However, high memory consumption is considered as the main problem of this technique.

Figure 9 shows how this approach works over time. If we assume we have N fixed sized chunks and the user's network speed changes dynamically over time, different levels of

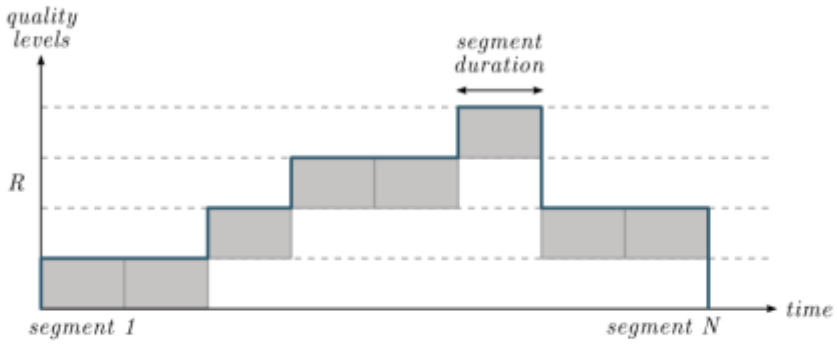

Figure 9. How different levels are served in stream switching [9]

videos are selected based on the available bandwidth and CPU capability. However, during one segment only one level is served and changes of levels happen only between the segments.

2.3 HTTP-Based Adaptive Bitrate Streaming

The HTTP-Based Adaptive streaming is based on two main bases, HTTP and adaptiveness.

By selecting HTTP as the communication protocol, we can benefit from using CDNs. A CDN helps by offloading the original server and improves scalability, coverage domain speed, and cost reduction.

In HTTP based adaptive bitrate streaming, there is no need to setup special media servers which are more expensive and the software architect can depend on current HTTP servers which are cheaper and easier to maintain. In addition, firewalls do not cause any unneeded blocking which was a problem in cases such as RTSP. The reason behind the choice of HTTP as summarized below[1].

1. HTTP streaming is spreading and known as the main form of delivering internet video.
2. HTTP matches with the idea of open internet especially for multimedia.
3. Firewalls will not block HTTP and network access address (NAT) can be avoided.
4. Simplicity of HTTP and TCP/IP deployment due to its wide use.
5. HTTP-based delivery is cost-friendly because standard HTTP servers and HTTP caches are cheaper than media servers.
6. It moves the control of the “streaming session” to the client. The client can open TCP connections which some of them might be in the cache.
7. The client can choose the initial quality level based on the bandwidth or user’s personal preference.
8. It facilitates the change of quality level automatically and without any interactions from the server. It is seamless to the client as well.
9. It enables the use of CDNs to increase delivery speed.

A conventional way to implement adaptive streaming is to stream over HTTP protocol. By using HTTP, the implementer can benefit from the existence of the current infrastructure of delivering web content to deliver HTTP-Based adaptive streaming.

The server usually keeps multiple encoded bitrates of a video and it serves each HTTP client according to the condition of the client. If the condition of a client changes, for example, its internet speed increases or decreases, the server adapts the sending bitrate

set to the proper one. In HTTP-based adaptive bitrate, video and audio sources are segmented in short pieces of roughly same size. Each segment is required to have a key frame in the beginning. In other words, first frame does not have any dependencies with its previous or next frames.

Playback starts in an order, meaning the client request first segment. Each segment is behaved like *progressive download* approach.

2.3.1 Microsoft Live Smooth Streaming (LSS)

Microsoft's smooth streaming protocol deliver live and on-demand media via HTTP. It uses MPEG-4 to encode the digital media which provides a seamless switching of bitrates in a near-real-time across different bitrates[12].

In an easy setup where there is no proxy or CDN, the client first asks for a manifest file. A manifest file is a metadata about the available bitrates. After getting the manifest file, the client first asks for the first fragment and after getting the response, the client asks for the next one until the media finishes or the client stops requesting the fragments. The figure 10 has depicted the process.

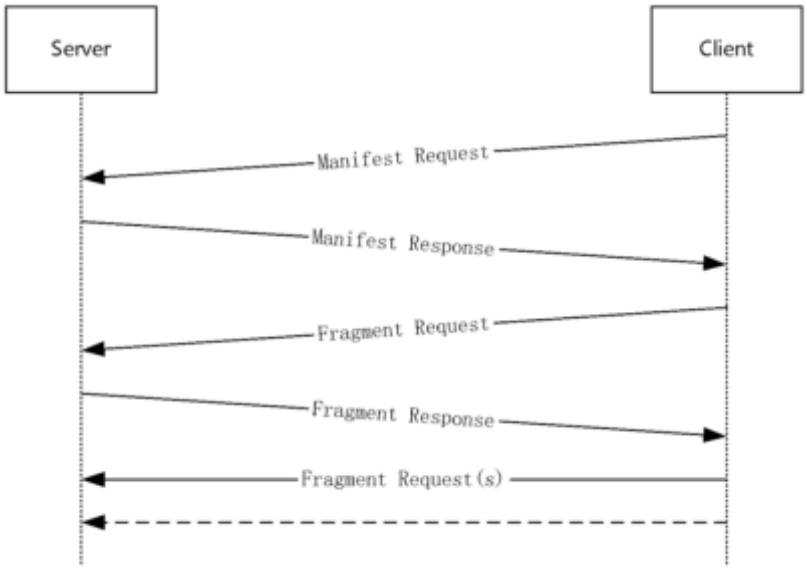

Figure 10. Typical communication sequence for the Smooth Streaming Transport Protocol [12]

Correlation between requests and responses are handled by HTTP.

The server in this architecture is stateless, and this enables us to use multiple instances of server like CDNs or HTTP cache proxies. Figure 11 shows the communication pattern of one fragment if we have a HTTP cache proxy.

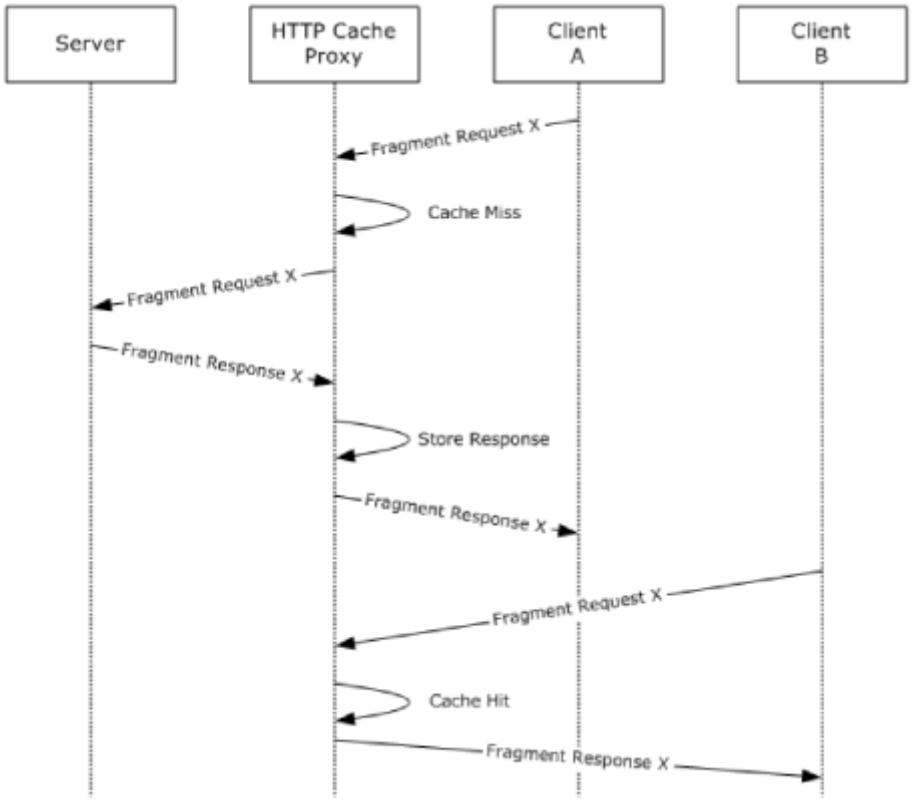

Figure 11. Typical communication pattern of requests for the same fragment [12]

Like every other protocol, Microsoft has defined the protocol which servers and clients must comply to. To get the manifest file the client must send a HTTP GET request to the server. The request must contain a **ManfiestRequest** header field. This way, the server can detect request type and sends the manifest response, in response to the request. The request to get the fragments must be a HTTP GET request which contains a **FragmentRequest** field. The manifest file must be a well-formed XML document which has the root element **SmoothStreamingMedia**. The following code fragment is an example of a simple root element of a manifest containing required and optional attributes[13].

```
<SmoothStreamingMedia
      MajorVersion="0"
      MinorVersion="0"
      Duration="0"
      TimeScale="0"
      IsLive="0"
      LookaheadCount="0"
      DVRWindowLength="0">
</SmoothStreamingMedia>
```

Table 1. Attributes of a root element in manifest file in Microsoft Smooth Streaming[13]

Attribute	Description
MajorVersion	(Required) Specifies the Client Manifest Major Version.
MinorVersion	(Required)Specifies the Client Manifest Minor Version.
TimeScale	(Optional) Specifies timescale for the entire presentation as several units that pass in one second. The recommended value is 10,000,000 which maps to increments of 100ns. 10,000,000 is the default if this attribute is omitted.
Duration	(Required) Specifies overall presentation duration of the media in increments of the TimeScale attribute. Duration should be set to 0 for live presentations whose approximate duration is not known in advance.
IsLive	(Optional) Specifies when the attribute value is, "**true**" (case-insensitive) that this manifest describes a live presentation that is still in progress. When this attribute is set to "**true**" the duration is interpreted as an approximation, and it is permissible for the presentation to end before or after the expected duration (for instance, if a sports event goes into "overtime"). For On-Demand presentations, this attribute should be omitted.
LookaheadCount	(Optional) Specifies the number of fragments in a lookahead.
DVRWindowLength	(Optional) Specifies the length of the trailing window for a 24/7 broadcast.

The root element can contain two child elements: **StreamIndex** which specifies the metadata for one type of track such as audio or video and **Protection** element which is a container for content protection playback info[13]. The following code is an example of StreamIndex:

```
<StreamIndex
  Type="video"
  Chunks="88"
  QualityLevels="8"
  MaxWidth="848"
  MaxHeight="476"
  DisplayWidth="848"
  DisplayHeight="476"
  Url="QualityLevels({bitrate})/Fragments(video={start time})">
</StreamIndex>
```

Table 2. Attributes of StreamIndex element in manifest file in Microsoft Smooth Streaming[13]

Element	Description
Type	Specifies the track type, and MUST be "audio", "video" or "text".
Chunks	Specifies the number of data chunks in the stream.
QualityLevels	Specifies the number of tracks for variable bit rates.
MaxWidth	The maximum width for video display.
MaxHeight	The maximum height for the video display.
DisplayWidth	The recorded width of video display.
DisplayHeight	The recorded width of video display.
Url	Specifies the format of **QualityLevel** (track) identifiers.

StreamIndex element can contain two child elements: **QualityLevel** which is a track for one of several bitrates needed for IIS Smooth Streaming media and **c** element which is a chunk identifier for segment of data[13].

In Microsoft smooth streaming, chunks are advised to be two seconds long.

2.3.2 Adobe HTTP Dynamic Streaming

HTTP dynamic streaming known as HDS is Adobe's implementation of HTTP based adaptive bitrate streaming. It supports videos encoded in H.264 or VP6. VP6 is Adobe's proprietary lossy video compression format and video codec. The workflow of this implementation includes content preparation tools, fragment MP4 files that are HTTP cache friendly, a playback framework and options for protected streaming powered by Flash access[14]. It supports both live and on-demand video content. It supports encryption of files to protect publicly available data. The playback is available only in Adobe Flash Player 10.1 onwards and Adobe AIR applications. This, enables different users with different platforms such as Linux, Mac OS and Windows to watch adaptive video on their platforms if they install Adobe's needed plugins.

The workflow of this implementation is depicted by Adobe's datasheet in figure 12. The workflow contains four levels: preparation, distribution, protection and consumption.

Preparation differs if the content is live or VOD. The output of preparation phase is to make media files which can be encrypted and manifest files in the server. F4F is the extension of MP4 atomized fragmented files, containing FLV packets. This format allows HTTP GET methods to fetch and cache smaller portions of the media. Once the data is stored on the server the client can get the files if it has the rights to access the media. Like other implementations, CDNs can reduce the load of the origin server.

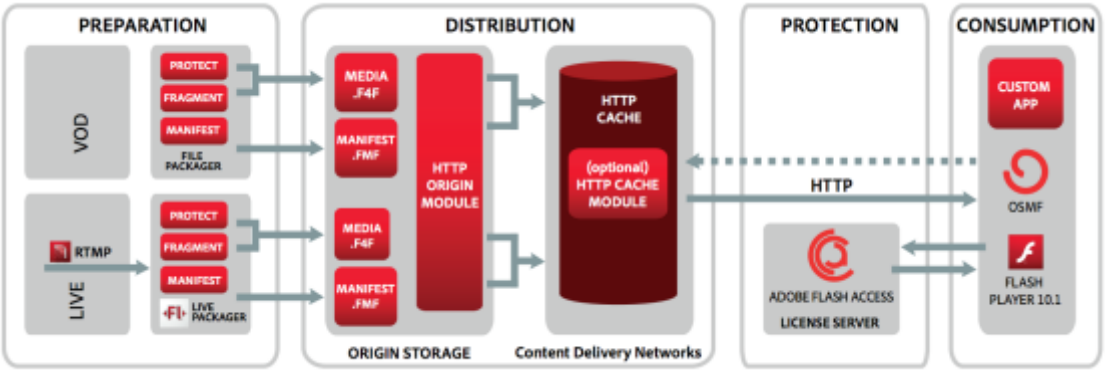

Figure 12. Adobe's HTTP Dynamic Streaming workflow

An example manifest file of HDS is shown in program 1. It is an XML-based meta data which gives information about existing on-demand video to the client. In this example, there are three available bitrates in the server.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns="http://ns.adobe.com/f4m/1.0" version="3.0">
    <id>myvideo</id>
    <duration>253</duration>
    <mimeType>video/x-flv</mimeType>
    <streamType>recorded</streamType>
    <baseURL>http://example.com</baseURL>
    <drmAdditionalHeader
      url="http://mydrmserver.com/mydrmadditionalheader"/>
      <bootstrapInfo profile="named" url="/mybootstrapinfo"
            fragmentDuration="4"/>
      <media url="/myvideo/low" bitrate="408" width="640" height="480"/>
      <media url="/myvideo/med" bitrate="908" width="800" height="600"/>
      <media url="/myvideo/hi" bitrate="1708" width="1920" height="1080"/>
</manifest>
```

Program 1. *Example of manifest file in HDS* [15].

2.3.3 Apple's HTTP Live Streaming (HLS)

HTTP live streaming known as HLS is the Apple's implementation of adaptive bitrate streaming. It facilitates both VOD and live streaming. It allows authentication and

encryption of content which the publisher wants to encrypt. Although the technology and architecture of HLS is designed by apple, there is no need for any special server and any normal HTTP server can serve the files and the clients can use Linux, Mac OS or Windows. Safari has the built-in support for HLS but in other platforms, open source software such as VLC must be installed.

The advantage of this system is its simplicity in comparison to Adobe's HDS and Microsoft's smooth live streaming.

The HLS architecture overview is depicted in figure 13.

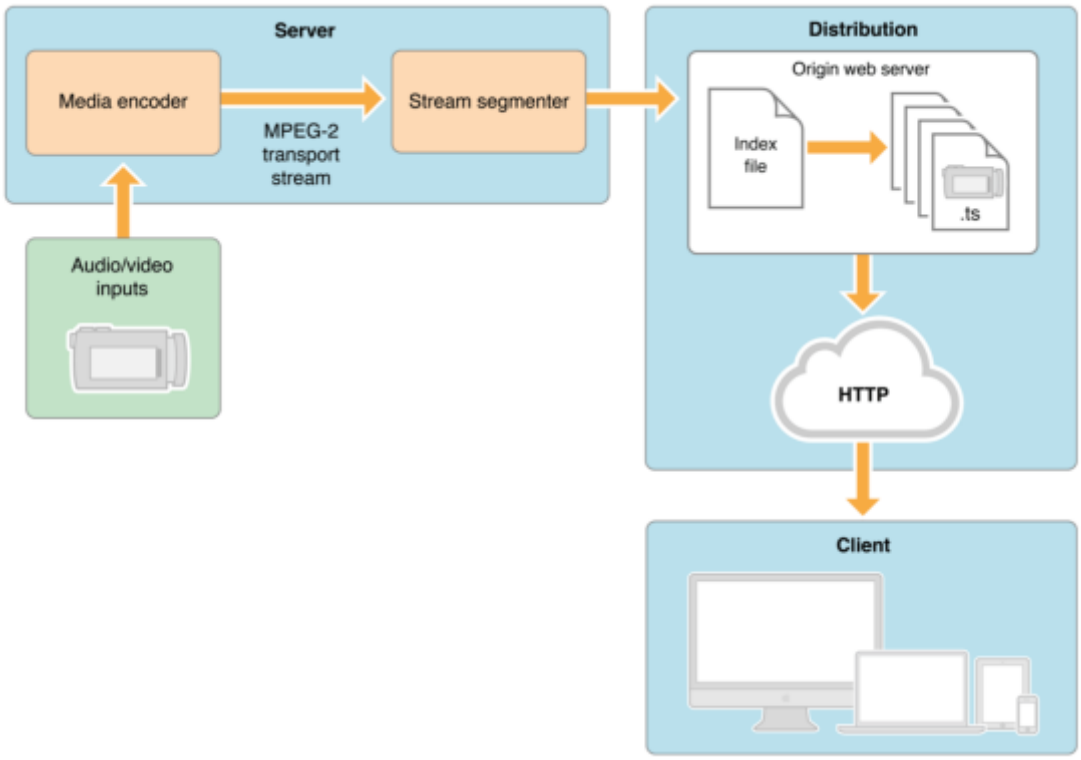

Figure 13. HLS architecture overview [16]

After getting the video from a file or a video capture device such as a camera the media encoder block encodes video files in H.264 format and encapsulates it by MPEG-2 transport stream containers with .ts extension. Afterwards, the stream segmenter divides the video file into almost equal sizes. The segmenter can be open-source applications such as FFMPEG or tools offered by Apple to do so. The chunks are advised to be around 10 seconds for optimal performance. Every chunk should be an independent decodable video file. Hence, they should all start by a Key-Frame. This is the reason why they cannot be exactly of the same size.

Unlike Microsoft Smooth Streaming and Adobe HTTP Live Streaming, the manifest file known as playlist is not an XML based text file. It is a text file with the extension of M3U8. The 8 letter at the end denotes that the text file should be encoded by UTF-8.

There are two types of playlists: master playlist and media playlist, both are UTF-8 text files containing URIs and descriptive tags. [17]. The relationship between the playlists is shown in figure 14.

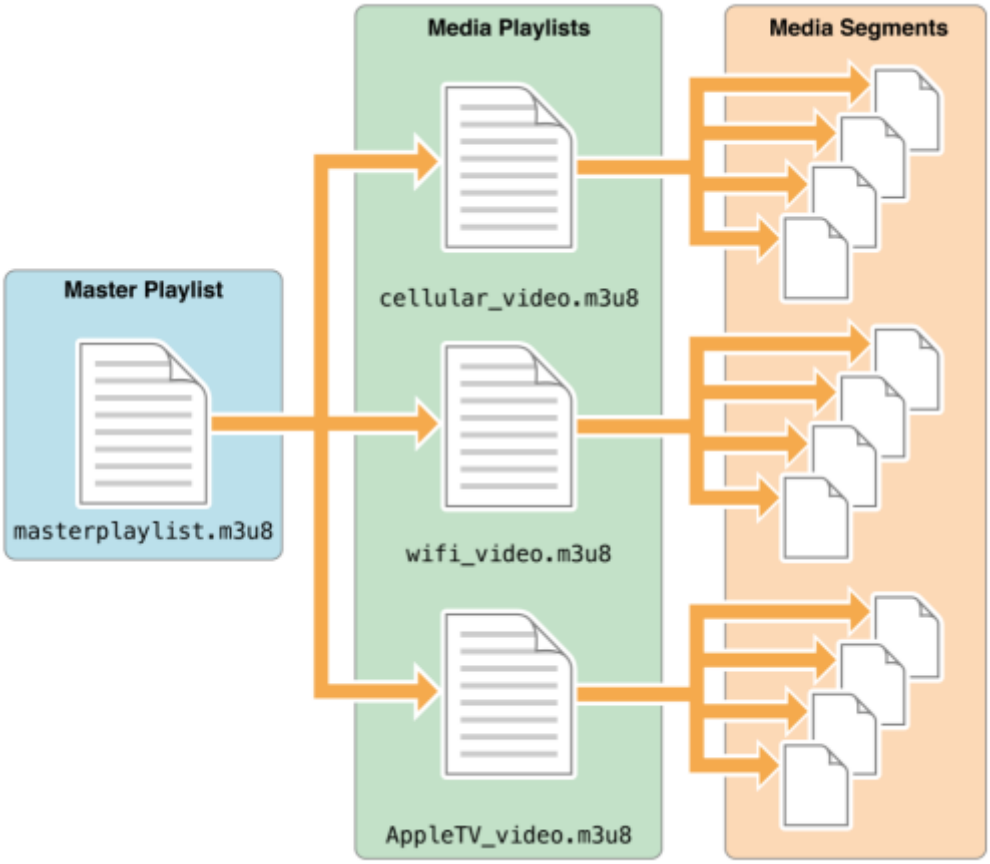

Figure 14. HLS playlist relationships [18]

The master playlist contains URIs for available encoded bitrates playlists. In the following example the master playlist contains three different encoded bitrates ranging from low quality to high quality, it also contains audio-only playlists,

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1280000,AVERAGE-BANDWIDTH=1000000
http://example.com/low.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2560000,AVERAGE-BANDWIDTH=2000000
http://example.com/mid.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=7680000,AVERAGE-BANDWIDTH=6000000
http://example.com/hi.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5"
http://example.com/audio-only.m3u8
```

#EXT-S-STREAM-INF is a tag which tells the client information about one available encoded set. The main attribute of this tag is bandwidth. Its value is a decimal-integer of bits per second. It represents the peak segment bit rate of the variant stream [17].

A media playlist contains URIs for media segments of same bit-rate encoded video. It can be a live media playlist which does not have any ending tags or it can be a VOD type with duration and seekability. If the streaming content is live, the playlist must get reloaded on an interval to load the new available URIs. The master playlist does not need to be uploaded.

```
#EXTM3U
#EXT-X-TARGETDURATION:10

#EXTINF:9.009,
http://media.example.com/first.ts
#EXTINF:9.009,
http://media.example.com/second.ts
#EXTINF:3.003,
http:/ /media.example.com/third.ts
```

Since HLS uses HTTP, proxies and CDNs can be used to increase the performance and decrease the work load on the original server.

Media segments can be encrypted using an advanced encryption standard known as AES. EXT-X-KEY is the tag in which its value tells if there is an encryption for the media segment. This way HLS can be guaranteed to provide a DRM system.

The client component is the part which wants to view the stream. First, it asks for the playlist. Then based on its network speed it fetches the suitable media playlist and buffer segments and if there is an encryption it decrypts them based on the URI for the keys. If the network speed decreases or increases, it may change the media playlist. The process continues until the client encounters the #EXT-X-ENDLIST [16].

HLS facilitates advertisement insertion through discontinuity tags. This is highly valuable because the common business model of online video content providers is by getting advertisements from companies and shows them at the beginning, middle or end of a video. Using the MPEG transport system brings more features to HLS such as closed captions and subtitles.

2.3.4 MPEG Dynamic Adaptive Streaming (MPEG-DASH)

MPEG-DASH was set-off in 2010 to bring the best rendered possible video streaming. Despite its late release in comparison to other similar technologies it has been used by major video sharing centers such as YouTube.

MPEG-DASH is also built upon HTTP and TCP based connections. TCP checks the data with error-checking to guarantee the received data is the same as the sent data. Although, TCP brings reliable data transfer service unlike UDP, video cannot completely rely on it due to the latency of error-checking, the user may experience media in which the video and audio are out of sync. In addition, in a normal TCP based connection a video must be downloaded completely before it can be viewed. These are the reasons which facilitate the need to have a framework upon TCP which guarantees the best possible user experience.

Although the previous and common implementations of adaptive streaming have been working well so far, the fact that MPEG-DASH is not limited to a certain company and it is not limited to proprietary underlying protocols and processes makes it as a widely acceptable ISO standard (ISO/IEC 23009-1) which all common streaming services have contributed to it.

As mentioned before, DASH is not a protocol or framework. It is an ISO standard which is a component of end-to-end services. It is codec-agnostic and the best possible codec can be used with it. An example architecture is depicted on figure 15.

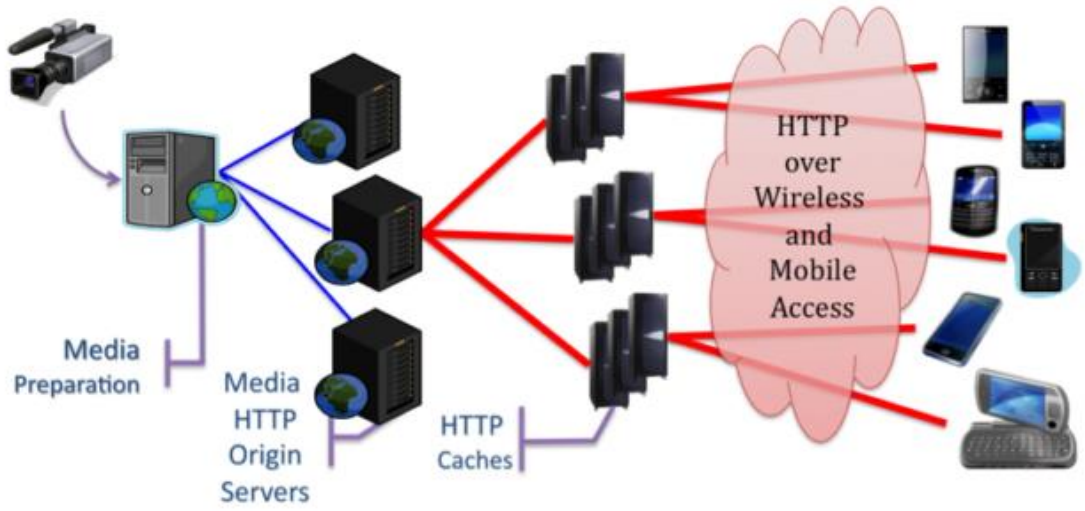

Figure 15. Example system for DASH components [1]

The modular concept of DASH enables the reuse of existing technologies such as DRM, encoders, decoders, etc. Due to the HTTP-Based concept of DASH, deploying it on top of HTTP-CDNs are also feasible.

MPEG-DASH's main goal like all other adaptive streaming methods is to deliver video content to clients over internet in an adaptive mode. It means the quality of the delivered video should be adapted to the client's internet connection status and other factors such as CPU capabilities. However, the client's internet speed is not stable during the playback of a video. The user may even for example switch from home network to his

mobile data system during the playback. Because the client is aware of its condition at any time, DASH allows the client to choose which of the available media sets to get from the server.

Media preparation module's task is to generate different sets of segments with different quality levels within the server. In live cases, it gets the media input from a camera or other capturing devices. In VOD cases, it gets the input from already available media files. The number of needed quality levels, codecs, duration of segments and the exact resolutions are optional and based on the need and capabilities of server and the client and factors such as cost can be decided by the owner. The media preparation module must also generate media presentation description (MPD).

The client first gets the MPD metadata from the server. The client selects the most suitable available version based on the MPD and client status factors. These factors include:

- Client Internet speed for selecting the quality level.
- Client processing and decoding capabilities.
- Locale setting for selecting a suitable audio or subtitles Language.

Based on the factors mentioned above, the MPD client sends the customized HTTP GET to the server. It is the client's task to switch to other quality levels based on the preference changes which might be asked for by the user, such as asking for subtitles, or which are caused by change of the situation in which the first segment was requested, such as the change in internet speed.

To get a better insight to the way DASH clients interact with a server or CDN, figure 16 depicts the conceptual modules and interactions within server and client.

Servers keep segments of media. Segment is defined by ISO as: "unit of data associated with an HTTP-URL and optionally a byte range that is specified by an MPD" and a media segment is defined as: a segment that complies with media format in use and enables playback when combined with zero or more preceding segments, and an Initialization Segment (if any)[19].

Server oversees sending MPD to the client and based on the incoming requests from the HTTP streaming client, sends corresponding media segments. Every HTTP GET request oversees one media segment.

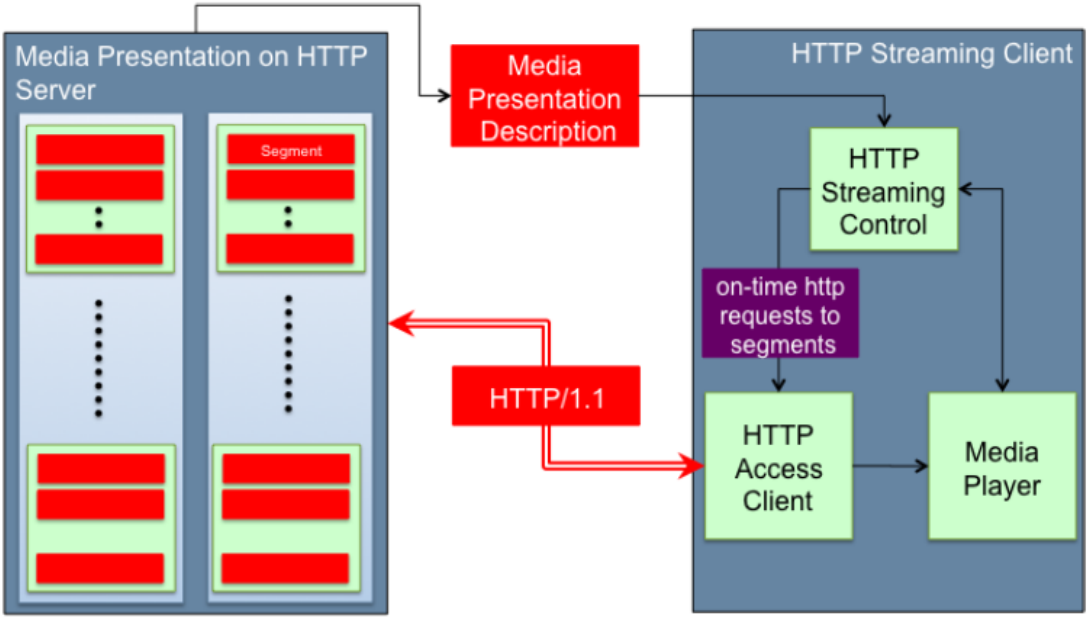

Figure 16. DASH data model [1]

On the other side, HTTP streaming clients consist of three main modules: HTTP streaming control, HTTP Access client and media player. The HTTP Streaming control module oversees parsing the MPD file and, based on the user preference and available bandwidth and CPU, decides from the available media sets in the server to send a request. The HTTP Access client is responsible for getting the command from the control module and sends HTTP requests to the server. The media player module must show the media. It gets the commands from the HTTP streaming control module. The control module decides when to play and when to pause to buffer ahead if needed. In case of user interactions such as pause, stop or seek, the control module first gets the request from the media player and then sends corresponding commands to both the media player and the HTTP access client module. That is the reason the line between these modules are bi-directional.

Media presentation description is a document than contains metadata required by a DASH Client to construct appropriate HTTP-URLs to access segments and to provide the streaming service to the user [19]. MPD is an XML-based document. The major elements of MPD and its hierarchy is explained in detail [1]:

- MPD consists of a series of one or more consecutive non-overlapping Periods.
- Each Period contains one or more Representation.
- Each Representation consists of one or more Segments.
- Segments contain media data and/or metadata to decode and present in the included media content.

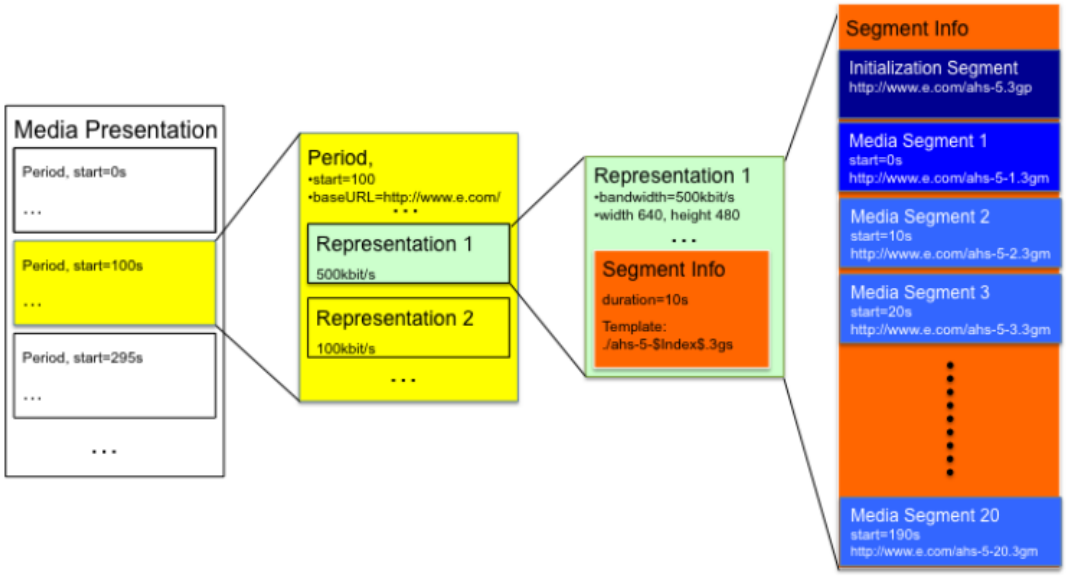

Figure 17. Media presentation data model [1]

Figure 17 depicts the mentioned hierarchy. In the root of the MPD, there are one or more Periods. By using the concept of Period, the system can make dynamic adaptive streaming happen. Period's start time are absolute times from the media. Each period consists of a base-URL. Periods consist of one or more Representation which makes the client aware of different available bitrates and quality levels. Representations are not bound to bit-rates only, some attributes such as language or subtitles language results in different representations. Each Representation consists of one or more media components where each media component refers to one media type such as video, audio, timed text for subtitles etc. Each media component comprises a group of media segments and at most one Initialization segment. Media segments are the smallest unit of media files which can be decoded independently.

The URL to a media segment can be built from the base-URL available at every period and the base-URL must be appended to attributes of each representations and media segments.

A MDP is a static XML document in case of VOD. However, in a live streaming scenario, it gets updated based on an interval. In a live case, there is a flow of capturing, encoding, potential transcoding, and then the MPD updates accordingly. By the update of MPD, the client is made aware of the latest ready parts of the media to play. Although the flow might be done in a rapid way, the live content has always a delay due to the needed process. To ensure the synchronization between the client and server, the MPD provides access information in Universal Time Clock(UTC) time. As long as server and client are synchronized to UTC time, the synchronization between client and server can be ensured by the use of UTC times in the MPD [1]. Program 2 is an example of a simple MPD XML document.

```
<MPD                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mpeg:mpegB:schema:DASH:MPD:2011"
xsi:schemaLocation="urn:mpeg:mpegB:schema:DASH:MPD:2011 dash.xsd"
minBufferTime="PT10.00S"
baseURL="http://www.example.com/"
type="Live" availabilityStartTime="2001-12-17T09:40:57Z">
<Period start="PT0S">
 <Representation mimeType="video/mp4;
  codecs=avc1.644028, svc1" group="1" width="320" height="240"
  frameRate="15" id="tag0" bandwidth="128000">
     <SegmentInfo duration="PT10.00S">
         <InitialisationSegmentURL sourceURL="seg-s-init.mp4" />
              <Url sourceURL="seg-s1-128k-1.mp4" />
              <Url sourceURL="seg-s1-128k-2.mp4" />
              <Url sourceURL="seg-s1-128k-3.mp4" />
     </SegmentInfo>
   </Representation>
  </Period>
</MPD>
```

Program 2. *MPD Manifest example*

Because the segments are stored in the server, time-shift viewing and similar activities are possible with DASH.

2.3.5 Comparison of the Technologies

There are many features we can evaluate to make a comparison between different technologies and implementations of adaptive HTTP-streaming.

Table 3 has provided a clear comparison between the different common implementations of different streaming services. It has compared Adobe HTTP Dynamic Streaming (HDS), Apple HTTP Live Streaming (HLS), Microsoft Smooth Streaming and MPEG-DASH based on features which are decisive in choosing a technology as a baseline. Deployment on Ordinary HTTP servers tells if the method can run on ordinary servers or if it needs special media servers. It also discusses if the method is codec agnostic or it needs a certain codec. End-to-end latency plays a major role where having the stream as early as possible in live events is favored such as sport matches or concerts. Other features such as favored length size of video chunks and maximum Supported Bitrates are also presented in the table.

Table 3. Comparison of different HTTP streaming methods

Feature	Adobe HDS	Apple HLS	MS Smooth Streaming	MPEG-DASH
Deployment on Ordinary HTTP servers	No	Yes	No	Yes
Official International Standard	No	No	No	Yes
Streaming Mode	Live/ On Demand	Live/ On Demand	Live/ On Demand	Live/ On Demand
Adaptive Bitrates	Yes	Yes	Yes	Yes
Media Container	MPEG 4 - Part 14, FLV	MPEG-2 TS	MPEG 4 – Part 14	3GPP, MPEG 4
Video Codec	H.264	H.264 Baseline Level	Codec Agnostic	Codec Agnostic
Default fragment length	4 seconds	10 seconds	2 seconds	Optional
Maximum Bitrate	Unlimited	1.6 Mbps	Unlimited	Unlimited
Multiple Audio Channels	No	Yes	Yes	Yes
HTML 5 Support	No	No	No	Yes
End-to-End Latency	6 sec	30 sec	>1.5 sec	flexible

3. DESIGN AND IMPLEMENTATION

This thesis suggests two complementary solutions that in the media streaming ecosystems do not exist. First In 3.1, Use cases are discussed. Afterwards, making several playlists and movie highlights out of one video source is discussed in 3.2.1, Enhancing and utilization of the network bandwidth for the made playlist is the complementary issue which is discussed in part 3.2.2.

3.1 Use Cases

The idea of both having bookmarks for media contents, and enhancing seeking towards them, are generic and they are not bound necessarily to a specific use case or platform. However, they are different in other factors such as scalability.

Regarding video bookmarking, if assuming video bookmarks are generated and accessing them is the only issue, if it is online, there is no setup needed in any client, and only developing a player which can get a bookmark, JSON file is enough. The development of this player has been done in desktop in this thesis, and the same logic can apply to players over the web. If it is not an online media player, the current player, which is a proof of concept, is cross platform and the same logic can be re-used to get a JSON file, like the subtitles are added to movies, it can be via drag and drop or other similar methods.

However, enhancing over bookmarks is not as scalable as bookmarks. This is because bookmarking is just a seek to a moment in video which has additional meta data, but enhancing seeking towards them requires caching, and depending if it is online or offline it may also require special privacy matters if the data must be secured. Although it is not as scalable as bookmarking, it can be used as a generic tool and more specifically, in cases where high quality movie must be stored, and there are different recorded videos from different angles. These cases are like CCTV and medical cases. Also, in cases where seeks to a media takes more time than normal due to low internet speed, it is also an advisable option.

3.2 Overall Design

3.2.1 Bookmarking

Video bookmarking is defined as an alternative way to make a video summary. Video summarization is a technique in which a shorter video is made from a longer superset. It can be a manual, semi-auto or fully automated process. There are plenty of use cases for why a video summary is needed. It can be highlights of a sports game such as a football highlights which is suitable for busy sport enthusiasts or it can be practical cases such as movements under the domain of a surveillance camera.

Video summarization cuts the video into several pieces and merges those parts together either with a transition effect, which makes the transition of smaller parts smooth, or without a transition. The output of such processes is a shorter video, which may meet the target consumer needs or not. If the viewer is satisfied with the summarized video, the summarization process was successful. However, the viewer does not share a similar interest level on the same topic. For instance, a fan of a football club who has missed a game, might want to see a longer video summary of the game in comparison to an average fan. It also happens with automated video summarizations. As stated in chapter 2.1.2, there are many techniques which are used to make a video summary. Taking motion based video summarization technique as an example, it uses a threshold, for the movement level to skip not interesting movements such as the flying of an insect. But the result might not be enough for the police and he or she might be interested in some seconds before the segmented video. This case and many similar cases, can be solved if the source video exists. Although it requires time and effort to get the missing part in the summary video, the data is not yet discarded and can be retrieved. The main problem occurs when the longer original video is discarded and data loss does not let the interested viewer reach the wanted part. This happens a lot due to saving disc space with regular cycles. For example, based on the settings, the recorded material of a CCTV camera might get deleted after a month.

On the other hand, it can happen a lot that different users make different summarizations out of a same incident. For instance, a football game, a movie, a concert or any other popular event.

Video bookmarking unlike video summarization suggests keeping the original video and making a standard event array out of it. It requires having an event media player which accepts the video and the optional event array as an input. Video bookmarking introduces and defines following terms:

Event: An event is a continuous span of media with content that makes it interesting to the video summarizer. An Event includes four attributes, start-time, end-time, duration and caption name.

Start-time: Start-time is an amount of time in milliseconds when an event starts. This is relative to the beginning of a media.

End-time: End-time is an amount of time in milliseconds when an event ends. This is relative to the beginning of a media.

Duration: Duration is the amount of time that an event lasts in milliseconds.

Caption: Caption is the meta-data about an event describing why an event has been highlighted and selected to be an event. It is a string such as “Hit the goal post”.

Event Array: An array containing all the events including its attributes. End-time is not needed because it can be calculated by summation of start-time and duration.

Event Media Player: It is a media player which can accept an event array as an input and can play the media linearly or just the events consecutively. It can act as a normal media player and accepts files or streams. If the event array is loaded successfully, the user will see the events which exist in the event array. However, the original video exists and user can look for an arbitrary moment. Figure 18 shows the schematic design of the event media player.

Event Viewer/ Selector: It is a list in an event media player showing the events and their information such as captions. It also allows the user to select an event and plays it.

The event array is decided to be a JSON array, to avoid introducing a new syntax and a JSON array can serve the need quite well. Moreover, there are many tested JSON parser modules already available which increases reusability and less coding is needed. The JSON array includes array of events with three attributes: caption, startTime and duration.

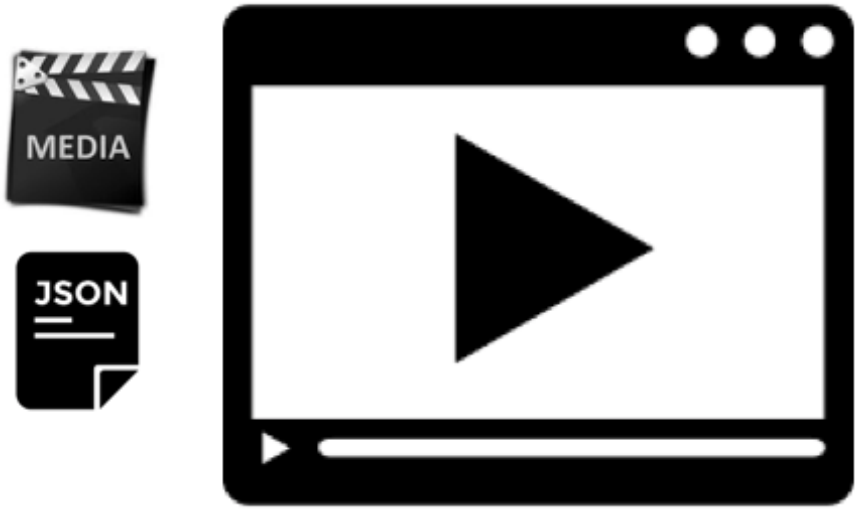

Figure 18. Event Media Player accepts Media file or stream and an optional event array.

```
[
{"category":"Goal Scored","startTime":3018266,"duration":50150},
{"category":"Goal Scored","startTime":4651000,"duration":20053},
{"category":"Hit The Post","startTime":2629423,"duration":14982}
]

```

Program 3. *Example JSON Array including three events*

Bookmarking a video has some pros and cons in comparison to conventional video summarizing. In the following paragraphs the technical comparison has been made between these methods:

The main advantage in video bookmarking is avoiding the data loss which occurs in video summarizing. Avoiding data loss has a high priority, especially in medical cases or security cases.

Another comparison factor between these strategies is consumed disk space. In this manner, neither method is superior over the other. In some cases, such summarized videos of CCTV with a motion based algorithm, assuming the algorithm has performed flawlessly, video summarization is better but as in chapter 2 it was discussed that in many adaptive streaming methodologies, several instances of media quality levels exist. In the server, it is different. In this case and similar cases bookmarking requires less space than summarization. In bookmarking only one JSON array is needed to make a video summary for all the quality levels. This is against the idea of making several summaries in different quality levels. Moreover, many people may make a video summary of the same material. For instance, for a soccer game, every sport news, related media or social media or even personal fans may make a specialized video

summary. In these cases, the total amount of needed storage exceeds the original video size. The idea of having one original video, and having several JSON arrays serve the same purpose, requires less disk space.

Requiring less time and processing in producing the video summary, in cases such as medical cases and security cameras, where several cameras record video from different angles is another advantage of bookmarking video. Because the videos are in sync and the same event array can be useful in other angles. It is true that processing each separately may result in more accurate summarization, but even in the worst-case scenario processing time is the same between these two methods.

An advantage of summarized videos is that they are easier to share and consume. This is because a player is not yet implemented, getting the event list as input. This is comparable to subtitle files. When they were introduced, most players, implemented getting subtitles as an optional input.

The main advantage of video summarization is reduced consumption of disc space. Video bookmarking suggests downscaling of non-eventful media parts in these cases, when we have very long and not so useful media content, such as captured CCTV. This way we can have both less required disk space and data loss avoidance.

3.2.2 Enhancing Seeking to Bookmarks

As mentioned in chapter ,1 in 2014 more than 65% of internet traffic is related to video content. This and similar statistics shows clearly that as time passes more and more video content are coming through the network, and less local files are getting used.

When a user watches a video online, based on the implemented technique, many scenarios might occur. In the most commonly used cases, which uses one HTTP based adaptive streaming, a similar experience happens to the user while watching the video.

Taking YouTube as the most commonly used video sharing website which uses MPEG-DASH as an example, the user first requests to watch a video. After detecting the status of the client, such as CPU capabilities and internet bandwidth, one of the available quality levels gets selected. The user must wait until adequate content gets buffered to avoid fluttery experience. Then the video gets played and if the status of the client does not change, switching between different quality levels does not happen. YouTube buffers the upcoming video ahead, and shows gray color to demonstrate the content which has been buffered. Figure 19 shows how YouTube informs the user about the point in the video that is being played and how much of the video has been buffered already. Formerly, YouTube was buffering until the end of the video. It was quite useful for users with lower internet bandwidth, letting them pause the video and wait for the video to buffer. However, YouTube changed this strategy because in many cases a

video was requested and then a new tab was opened, and the buffered content was left unseen. This strategy saved costs and lowered the unneeded traffic. The problem with users who had lower bandwidth, was also solved with the adaptiveness of YouTube. Seeking to the gray part happens without the need for buffering. Seeking to not-buffered parts, requires getting adequate content buffered which needs some time, based on the internet bandwidth.

Figure 19. YouTube buffers video ahead

Linear buffering ahead like other media streaming clients is perfect in cases such as YouTube, because most users watches the video linearly. If a user wants to see a special moment, he or she can seek and the buffering starts from that moment in media.

In the previous section, the idea of keeping the whole media and having a bookmark list was presented. It is apparent that the viewer is mostly keen on watching the events, although having access to the whole media is precious to the user. In an event media player client, linear buffering works but the user experience can be improved, if the expected seeking targets which are events are also buffered ahead, before the seeking takes place. Figure 20 shows the events with a yellow color in the media timeline. And red color as the current moment in the playback, and gray color as the buffered content. If the user selects an event in the event list, a seek request is sent to one of the yellow moments in time. As the buffering has happened linearly, the seek target is not yet buffered. This means waiting for getting adequate content, and it is not favorable because the user behavior was predictable.

Figure 20. Linear buffering(upper) shown in grey vs buffering based on the events (lower) which results in faster seeking to events.

3.3 Proof of Concept Implementation

Although the problem and the solutions are generic and the thesis role is to suggest alternative ways to summarize video, the proof of concept is done as a part of a Neuro Event Labs project. Neuro Event Labs Oy is a Finnish company which was established in 2015 as a spin-off of Intopalo Oy. The goal of Neuro Event Labs products is to help doctors in the treatment of epileptic patients. The diagnosis and treatment of epilepsy is performed more accurate and effective, if doctors know more about the seizures that their patients are suffering from. This is at its best when the doctor sees the patient, while they are having a seizure. Conventionally, the patients book a time from some specific hospitals which provide this service, they spend several nights sleeping in the laboratory for electroencephalography, also known as EEG, which monitors the electrical activity of the brain. The nurses are awake, and they monitor the patient and they mark the times when the patient starts and ends a seizure. A neurologist will check the patient recorded video beside the EEG activities, and the treatment decision is made based on that information. Afterwards, the scenario repeats based on an interval to see if the types of seizures, and their intensity has changed or not. This type of treatment is expensive, and the sleeping pattern changes in the hospital for the patient. Neuro Event Labs, also named as NEL, implemented a recording box which is easy to install in the patient's bedroom. The patient sleep is recorded by three cameras from different angles, which use night vision technology and the recording box, encodes the captured video, and stores it on the server. Afterwards, a motion based algorithm based on OpenCV, analyses the video, and marks the likely seizures known as events. Transcoding video into different quality levels also happens in the cloud based storage, so that adaptive streaming implementation can be done. After verification of credentials, the doctor gains access to patient data, and can watch the video and suggested events. The doctor can also mark new events. This whole process is significantly less expensive, and the algorithm performance can also get better and better, as the doctors input their opinion about the suggested events.

Another priority of NEL system over the current system, is the fact that NEL uses full HD videos while current system uses VGA quality. The better-quality video helps doctors to observe the seizures better. However, streaming three simultaneous Full HD video requires a fast Internet speed.

The recording box stores data when there is no Internet access, and can send the recorded media when it gets connected. Figure 21 depicts the schematic view of the design.

Figure 21. Schematic view of Neuro Event Labs Architecture

Proof of concept has been done over the NEL infrastructure. Hence, NEL infrastructure is explained in the following lines. The recording box is connected to a touch display, and uses a QT based application, allowing the patient to start or stop the recording. It encodes the video using h.264 with FFMPEG, and sends the chunked video data to AWS. For privacy purposes the data is encoded using the AES encryption technique. Every video chunk is 10 seconds long. When the last video chunk is sent to the server, the OpenCV Motion based algorithm starts processing each chunk. The output of the process is a JSON based file, which is the event list.

Apple HLS has been selected technique for streaming. The manifest playlist, which is an M3U8 file, is generated within the server. When the process is done, a neurologist can watch the patient data. In the client side, a QT based application is implemented, which shows the patient from three different angles. It also shows the event list. The client applications allow the doctor to switch on or off a special angle or swap the main view. An expandable side bar exists on the right side, which contains the event list. On the right side of the player, the normalized movement data from every camera is shown. Every event is shown as a highlight over the movement plots. On the top side of the application a timeline is shown. The timeline has a white expandable rectangle inside, which specifies which timespan of the recording night is shown.

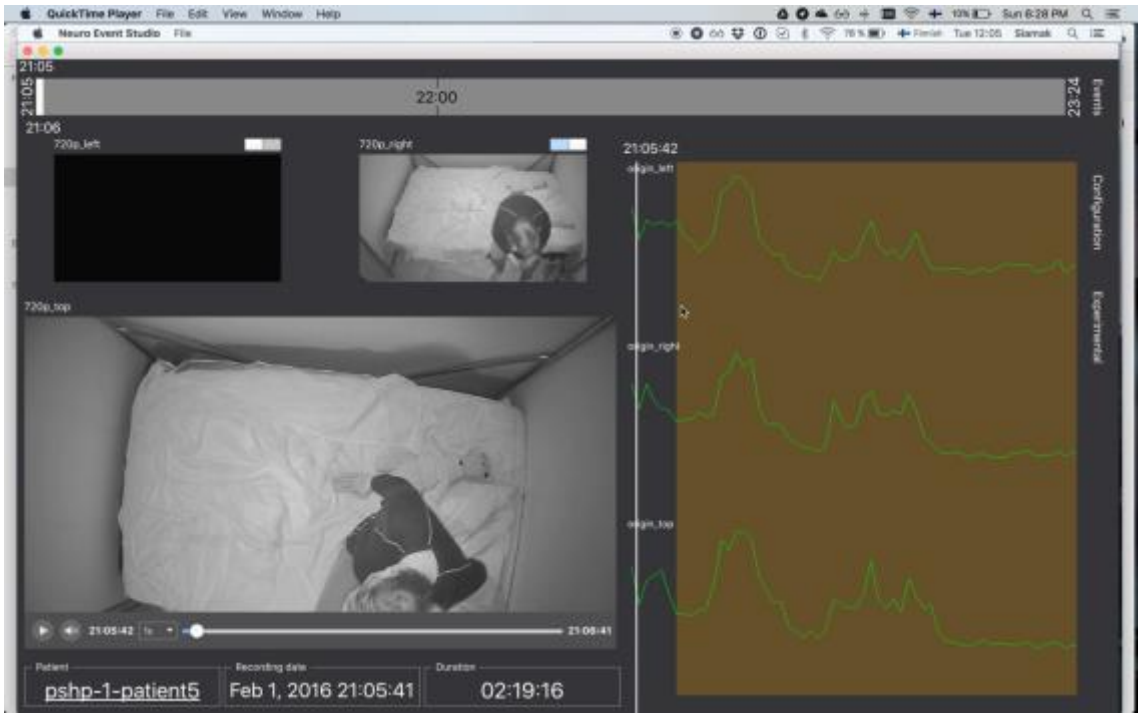

Figure 22. NEL QT based client application

Unlike figure 22, in figure 23 the event expandable tab is expanded. The neurologist can select an event to watch, tag, and mark it as a seizure or a false event detection. The feedback of a doctor, including tags or changes in the beginning or end of an event, improves the learning as it is counted as learning data for supervised learning.

Figure 23. An expandable event on the right side of the application can be tagged by a doctor

Implementation of proof of concept has been done over NEL infrastructure, and parallel to implementation of NEL itself, from January 2015 to June 2016 in a group of three. Qt 5.6 and QML are used for the client application and the recording display for the patient. Angular is used for patient selection after logging in. REST API has been used for the communication of both AWS and recording server, and client and AWS. JSON format has been used for event arrays. Python was used to perform OpenCV motion based algorithm in AWS. GIT is used for version control, and by integration of Bit Bucket and GIT code reviewing has been done. Scrum is used for agile software development. Sprint spans length were weekly. Bookmarking has been implemented completely in the software but enhancing seeking to bookmarks has been done after that separately as a proof of concept.

3.3.1 Bookmarking

After processing of video chunks, a JSON file like figure 24 is generated. Events can also be created by the neurologists on the client side. Every modification, insertion or deletion changes the JSON event array accordingly. The client first uses the network access manager to load the JSON file in the Qt client. Afterwards, it converts the JSON into known types in c++ and STL. Using QML objects, they are shown in a list view with features such as filtering, adding tags, removing tags, changing the status of an event, moving start time or the end time of an event, calculating duration, tagged icon for a delegate in the list view, etc.

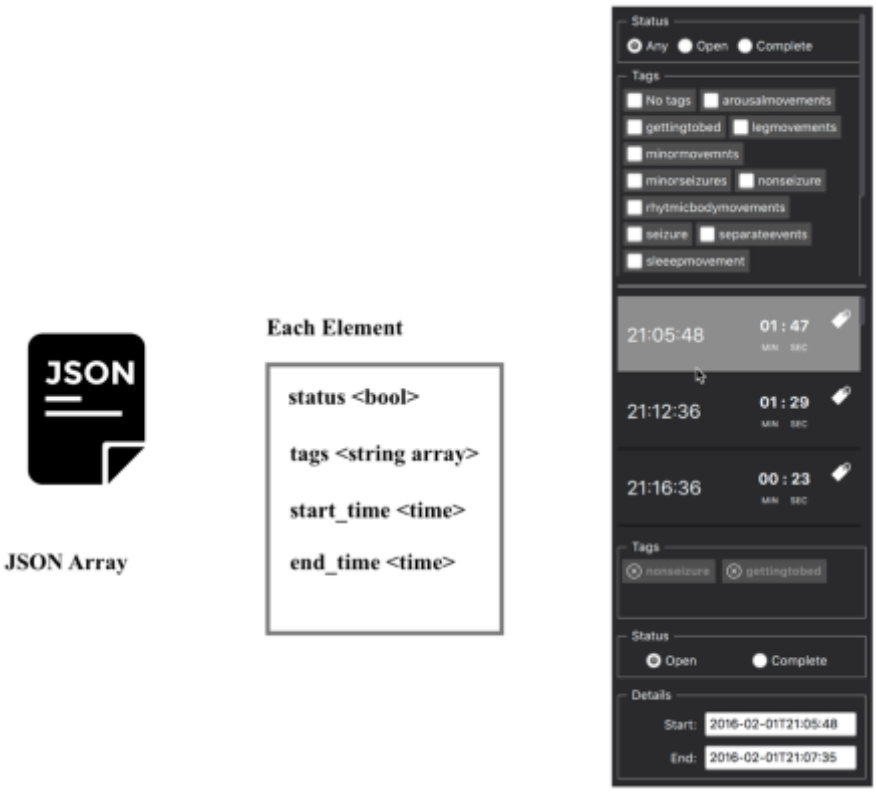

Figure 24. JSON Array is shown as an editable list view to the neurologist

After clicking every list item, the stream is seeking to the detected event. In the meanwhile, doctor still has access to other moments in the recorded material and can seek before or see more than the end of the selected event. Different doctors might alter the event list in their own way, meaning that different bookmarks can exist for the same media. This way, for different bookmark sets, only one instance of original media is adequate.

The bookmarks also represent themselves as faded rectangles over the motion graph, as shown in figures 22 and 23. The white cursor over motion graphs moves as the user plays the video, and when the white cursor hits the rectangle, it means the event has started. If an event is selected from the side bar, a seek is done to the start of an event, which means the white cursor starts again from the start of a rectangle. Based on the event duration, the rectangle's width is drawn meaning that when a rectangle is finished, the event has finished.

Additional requirements, other than just bookmarking, can be implemented by adding attributes to the JSON. For instance, for tagging and commenting, only optional JSON arrays of author and tag, or commenting content had to be added to the JSON structure, and sending the updates and parsing it back had to be implemented.

Although there exists a lot of JSON parsers, the parsing has been written from scratch to avoid making the application size bigger with unneeded methods, because it was only a handful of attributes which had to be parsed and converted to Qt data types.

Double clicking over a rectangle was causing the sidebar to get open for modification, this way the user did not need to open the sidebar and search for the event.

3.3.2 Enhancing Seeking

When bookmarks are available, it is predictable that the starts of the bookmarks are seek targets. Thus, buffering them ahead is useful for providing a better user experience for the user, when he or she clicks on an event.

Implementation of buffering events ahead depends on the streaming technology which has been used. In case of NEL, Apple HTTP live streaming has been used. Although chunk sizes are 10 seconds, the output M3U8 playlist does not have exact 10 second video chunks. FFMPEG or Apple's HLS tools generate the nearest possible duration video chunks, because every chunk must be independently decodable. Hence, every chunk must start with a key frame. In Figure 25, an example m3u8 file is shown. Although, in the Target duration the setting was requested to have video chunks of 11 seconds, the video chunks are not exact, and there is even a 5.4 second video chunk. The reason lies behind the fact that every chunk must be independently decoded, and they should start with a key frame.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:11
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a0.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a1.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a2.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a3.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a4.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a5.ts
#EXTINF:5.400000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a6.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a7.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a8.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a9.ts
#EXTINF:10.800000,
https://s3-eu-west-1.amazonaws.com/isdapi2/football/a10.ts
#EXTINF:10.800000,
```

Figure 25. Video chunks duration are not exact because they must start with a key frame

In a normal situation, the buffering is done based on the client implementation. Different clients can play HLS such as Safari, VLC etc. By inspecting network traffic caused by playing the stream, it turned out that they buffer two to four chunks ahead, and they start playing the stream when they have downloaded two to four chunks already. When a seek happens they calculate which video chunk includes the target seek, and they fetch the chunk. The calculation is feasible because the m3u8 includes exact media duration, and by summing the video chunks' duration, the relative start time and end time of every video chunk exists. In this case, every seek time equals the time which the client application receives the command, till it plays the target moment in video. The time approximately equals to the time in which the adequate video gets downloaded. Because decoding time and other times are not comparable to this amount, and this is counted as the bottleneck of seeking.

In the case of NEL, because there were three Full-HD cameras capturing patient from different angles to provide more data, seeking was done when all the videos had performed seek action successfully and ready to play. Hence, the seeking actions were not fast and the doctor had to wait after each seek.

To overcome the long seeking actions, and enhance seeking time, the application buffered the eventful chunks. They are in practice the start times of the bookmarks. Chunks which the user is more likely to request are prioritized.

An m3u8 parser class firsts parses the m3u8 file, and checks if it is a master playlist or a media playlist. If it is a master playlist it picks the first playlist, otherwise the media playlist is chosen. It first calculates each video chunk's relative start-time and duration, and keeps it.

When the bookmark JSON is downloaded, the application finds the start time and the duration of events. The output of this process is the list of video chunks that include some parts which exist in the bookmarks.

The client application final goal is to be able to both play the media normally and enable enhancing seeking to bookmarks. It reaches the goal by finding the eventful chunks, and buffering them ahead.

Finding eventful chunks as the figure 26 depicts is performed by processing the bookmarks JSON and playlist manifest, which is M3U8 file manifest in HLS.

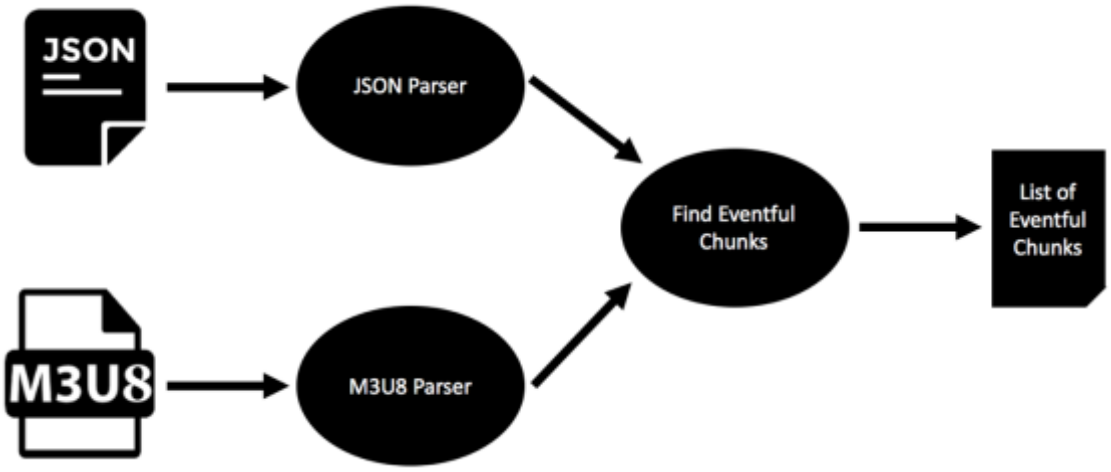

Figure 26. The process by which the list of eventful chunks is present

When the eventful chunks are known, they should be buffered and in case of request, the application should use the buffered chunk, instead of the file existing in remote chunk. This process is possible by designing a proxy over the network request. The proxy has a defined memory, and keeps the eventful chunks, if the request comes as predicted it is considered as a hit and the buffered video chunk is returned in a byte array as HTTP POST response to GET request. The schematic design is depicted in figure 27.

Depending on the cache size and the number of eventful chunks, the cache proxy may store all the video chunks, or may have some of them.

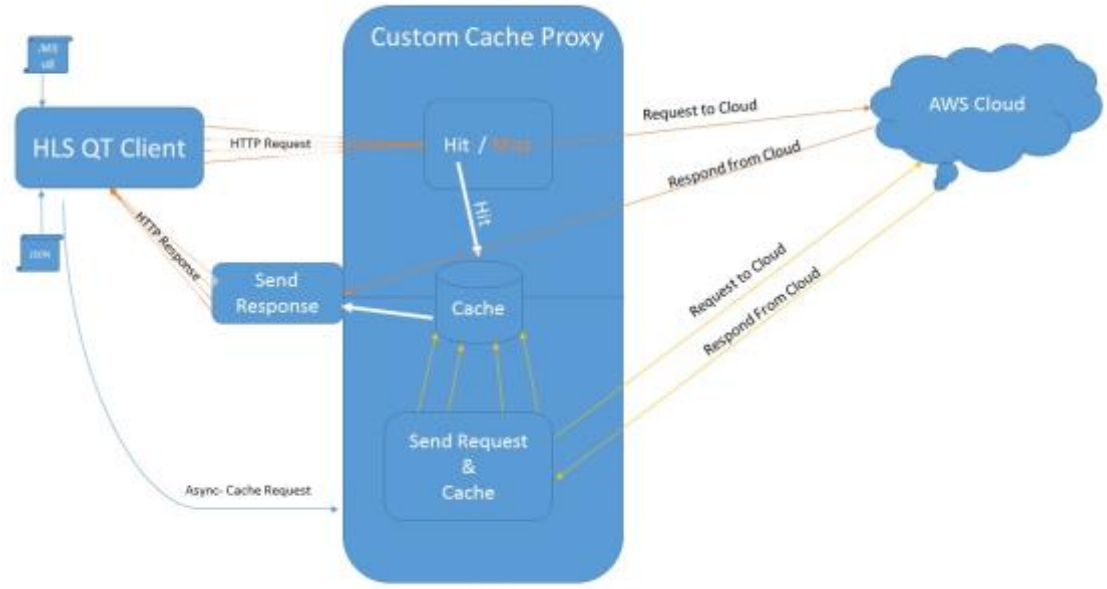

Figure 27. Schematic design of buffering cache

In the above figure, the HLS Qt client sends multiple HTTP requests as the playback begins. This is because the Qt Network Access Manager can send multiple network request to the server at a time, which askes for the first parts of the video chunks. Initially, all the requests miss the cache and the request is forwarded to remote server, which is the AWS client in this project. The response comes to the application which has requested which is the proxy. The proxy then forwards the reply as a reply to the original request.

In the meantime, the HTL Qt client also sends an asynchronous cache request to the proxy, and asks the proxy via a flag parameter to store them in the cache. This does not interfere with the normal behavior and playback of the media, and prepares the cache to store the needed chunks to increase HIT rate, which means enhanced seeking in general.

4. RESULTS AND EVALUATIONS

The present thesis project has aimed to improve two supplementary issues: Bookmarking and enhancing seeking towards them, in media especially over video.

Bookmarking goal has been totally met. The client player was supporting both videos without bookmarks and with bookmarks. In other words, the generic client player could be used both in normal cases, and when bookmarks are available. The client application uses a side bar to let the user decide to focus on the video, or watch the video while having the sidebar open.

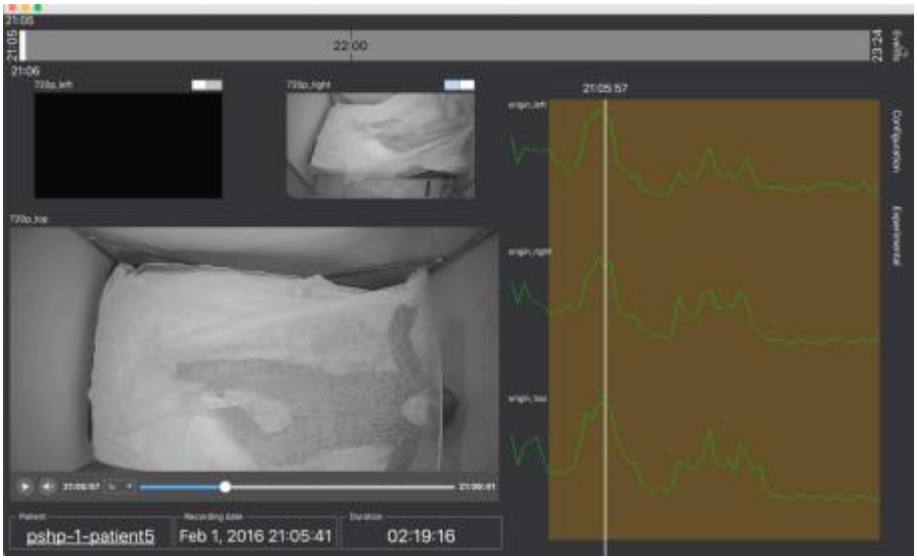

Figure 28. Bookmarks were on a sidebar that could be pinned in NEL they are called events. It can be expanded by clicking the button in the top right.

By customizing features over special cases in bookmarks, extra information could be achieved. For instance, in NEL every bookmark could take a list of tags. A user could also introduce a new tag. Bookmarks could be filtered by their tag. Although the goal of having bookmarks is met, and extra features have also been developed over it, commenting and showing or hiding bookmarks based on the creator of the bookmarks, could be the next step to further developing bookmarks.

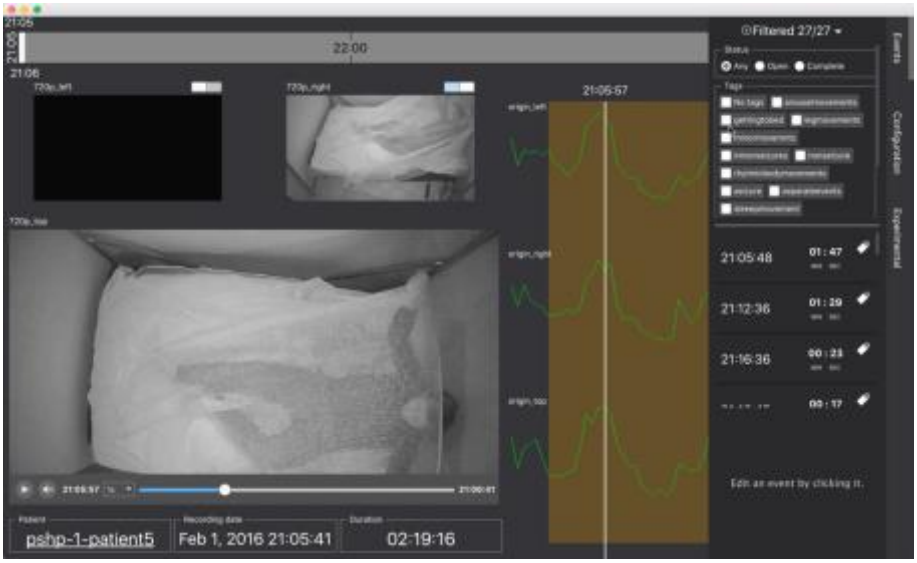

Figure 29. Extra features were also added to make bookmarks more efficient

The results in seeking towards bookmarks were measured in different cases. Making a seek command right after playing the video was having no result, as there had been no time to cache anything. However, depending on the internet speed, cache size and bookmark list, zero time for buffering the video has been achieved. This zero latency has been achieved because the video has been cached already, and no actual network request came out of the proxy layer. Currently, the proof of concept works in a way that it starts caching each eventful segment one by one, until it reaches the capacity limitation. As a result, if the first user selects the first item in the bookmark list after a couple of seconds, zero time is spent over buffering. If the eventful chunks do not exceed cache capacity after a certain moment in time, all seek actions takes zero time to buffer and they all have been cached before.

4.1 Limitation and Issues

4.1.1 Limitation of HLS

Developing the proof of concept over the NEL project was limiting development to work only over HLS. The proxy which caches the video chunks can develop in a way which it detects the streaming protocol, and calculates the eventful chunks and caches byte arrays or video chunks accordingly. However, HLS was used in NEL and because it is an Apple property, there is no built-in support for it in Windows. There is a need to setup other software to make HLS readable in Windows, such as K-lite codec pack, which has LAV filters included. Moreover, the seeks in Windows were not exact. It means in a Windows build, if the user was asking a seek command to moment 12000 in media, it could happen that the multimedia framework makes a seek to 12100. Although the difference was a minor time span, it makes a huge defect on the user experience

where we have 2 or more simultaneous videos, which go out-of-sync as the result of imperfect seek.

4.1.2 Limitation of encrypted data

In NEL infrastructure and many cases, the media content is private and it should be encrypted or event kept in a private server. HLS supports for AES encryption and the URLs can be toward every server, in which additional authentication can take place. In case of Bookmarks, authentication and encryption makes no difference. In terms of enhancing seeking toward bookmarks, the problem arises. In practice, while caching, the sensitive data is stored in the memory of the client device. Although there are means to make it secure, it is more likely that the security requirements for developing under this architecture increases.

Although security is inevitable especially in medical cases, having encryption and layers of authentications increases the accessibility time of the media, which is against the main goal of the project.

4.1.3 Generation of Bookmarks in NEL

Bookmarking media in this content has been done based on an OpenCV algorithm. Every user could add, edit or delete a bookmark. Other than needing to make a role definition, and access rights about who can make changes to what and if it is kept in a separate bookmark set, or if there is one copy that it changes based on the last edit. The limitation with generating bookmarks in NEL was that it was complex and different per patient. As a result, it required a lot of process over data, and it was not generic to include it as a part of thesis. Moreover, due to nature of NEL, there was not that much audio and text available. In case of other materials which have audio and text available, such as movies or sport matches, there could be a bookmark generator tool which based on the user's input, generates a bookmark set. For instance, in a soccer match, if a user wants to see the goals to make a video summary or to watch it, based on the commentator's audio, an automated bookmark generator could be developed. This tool could be generic and apply to other media. For instance, the user could search a sentence which he or she liked in a video and finds it.

In NEL case, the process of automating bookmarks was so complicated because every patient has a special pattern of seizures, which also changes between nights and a simple generator was not enough for finding a seizure. Moreover, a movement pattern in one specific part of the body, is decisive for marking it as a bookmark. For instance, limb stiffness accompanied by open eyes can be a seizure, while one of them alone may not be enough.

5. CONCLUSION

In the thesis project, an application has been developed in Qt as a supplement to NEL software. The final application consists of both the code from NEL team, which was a player for recorded epileptic patients, and my own development, which enhanced the seeking over bookmarks. Although it has worked flawlessly and with expected results, it is just a proof of concept because it is not secure to use it with patient data, which includes their private data.

The final application release was a disk image file operating in OS X, but it was cross platform because it has been written in Qt, and it could port to other platforms.

Due to different multimedia framework and lack of built-in support for HLS in Windows, seeking accuracy was not as good as OS X seeks. In Windows, based on the version, the multimedia framework is either DirectShow or Microsoft Media Foundation, and in OS X it is handled via AV Foundation which has built-in HLS implementation.

The product owner feedback about this plugin over their application was positive. However, they have had concerns regarding the security of cached data. Because there were strict requirements from hospitals about private patient data. However, it was acknowledged that the bookmarking and enhancing seek action to them is a generic idea, which can be used not only in NEL application, but also in every media which people might want to make a summary of. From a drama movie or a soccer match, to long CCTV recorded media or patient data.

Further developments can be done both in including bookmarking in players, and enhance seeking toward them. In bookmarking development currently, the idea is solid and further development is just adding features to it. Merging and integrating different bookmarks by different users, and offering it as an alternative, is one example of this kind. Moreover, allowing the users to view multiple bookmarks on the same view panel, is another way of improving the bookmarking idea. In enhancing seeking to bookmarks though, there is a lot of room for improvement. First and foremost, developing an application which works across most of the common online streaming methods, such as DASH, HLS, Microsoft Smooth streaming and Adobe’s dynamic stream. Detecting the type of the streaming can be easily done by the link or the manifest file. In addition, smarter ways of using the cache size are the next potential research study as a next step. Currently, the application stores as much as it can linearly from the eventful chunks.

However, it can act according to the user behavior. For instance, if the user is selecting one specific type of the bookmarks, for instance one special seizure, of bookmarks which have one similar tag, the cache can predict that the user is going to see similar things and start to fetch them or in more simple scenarios, if a user is watching one bookmark again and again, it can use a priority queue to keep and discard the space in a utilized manner. Protecting the proxy to be secure against unwanted access, is another way of further developing the proxy.

REFERENCES

[1] T. Stockhammer and Thomas, “Dynamic adaptive streaming over HTTP --,” in *Proceedings of the second annual ACM conference on Multimedia systems - MMSys ’11*, 2011, p. 133.

[2] 2016 Cisco VNI, “White paper: Cisco VNI Forecast and Methodology, 2015-2020 - Cisco,” 2016. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

[3] A. Rav-Acha, Y. Pritch, and S. Peleg, “Making a Long Video Short: Dynamic Video Synopsis,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR’06)*, 2006, vol. 1, pp. 435–441.

[4] T. Sebastian and J. J. Puthiyidam, “A Survey on Video Summarization Techniques,” *Int. J. Comput. Appl.*, vol. 132, no. 13, pp. 30–32, 2015.

[5] M. Ajmal, M. H. Ashraf, M. Shakir, Y. Abbas, and F. A. Shah, “Video Summarization: Techniques and Classification,” Springer Berlin Heidelberg, 2012, pp. 1–13.

[6] H. Sun, A. Vetro, and J. Xin, “An overview of scalable video streaming,” *Wirel. Commun. Mob. Comput.*, vol. 7, no. 2, pp. 159–172, Feb. 2007.

[7] C. Patrikakis, N. Papaoulakis, C. Stefanoudaki, and M. Nunes, “Streaming Content Wars: Download and Play Strikes Back,” Springer Berlin Heidelberg, 2010, pp. 218–226.

[8] H. Schulzrinne, “Real Time Streaming Protocol (RTSP).”

[9] L. Rubio Romero, “A Dynamic Adaptive HTTP Streaming Video Service for Google Android.”

[10] L. De Cicco, S. Mascolo, and V. Palmisano, “Feedback Control for Adaptive Live Video Streaming.”

[11] “ITU-T and ISO/IEC JTC1, H.264 and ISO/IEC 14 496-10 (MPEG-4) AVC Recommendation. Advanced video coding for generic audiovisual services. 2003.” .

[12] Microsoft, “[MS-SSTR]: Smooth Streaming Protocol,” pp. 1–67, 2016.

[13] Microsoft, “IIS Smooth Streaming Client Manifest - SmoothStreamingMedia Element,” 2008. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee673438(v=vs.90).aspx.

[14] Adobe, “HTTP Dynamic Streaming Datasheet.”

[15] Adobe, “Flash Media Manifest File Format Specification.”

[16] Apple Inc, “HTTP Live Streaming Overview,” 2016. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html.

[17] E. R. Pantos, “HTTP Live Streaming internet draft,” Sep. 2016.

[18] Apple Inc, “About HTTP Live Streaming,” 2014. [Online]. Available: https://developer.apple.com/library/content/referencelibrary/GettingStarted/AboutHTTPLiveStreaming/about/about.html.

[19] “ISO/IEC 23009-1:2014(en), Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.” [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:23009:-1:ed-2:v1:en. [Accessed: 25-Dec-2016].